Controlling FPGA Configuration with a Flash-Based Microcontroller

Introduction

SRAM-based FPGAs like the Atmel AT6000 series come more and more into use because of the many advantages they offer. Their reconfigurability allows the user to implement more gates in his application than the FPGA actually has, simply by loading the gates as needed into the FPGA. This is also called "Cache Logic™ ." For an efficient use of cache logic, the FPGA must meet the following requirements: partial reconfigurability, a fast reconfiguration process and full architectural symmetry.

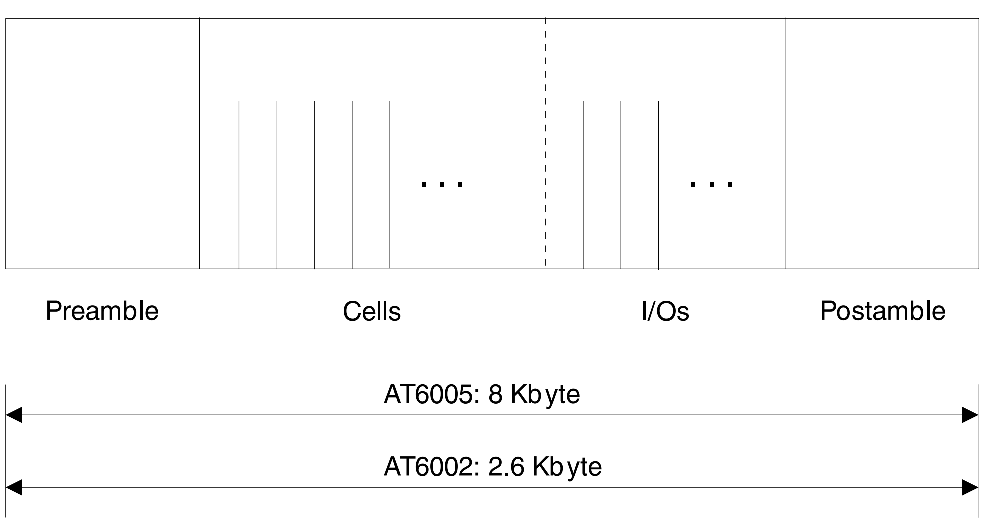
The FPGA can control and change its configuration itself, but this can also be done in a very elegant way by a microcontroller. After the configuration process or in-between two configuration cycles it can be used for other purposes and is not lost for the application. The different options for space-saving realization, design protection or for fast, flexible reconfiguration are shown in this application note. The microcontroller used here is the Atmel AT89C51 which is fully compatible to the industry standard i8031.

Configuration Data Transfer between the FPGA and the Microcontroller

The amount of information that makes up the configuration information for the FPGA is called a bitstream. It is a file stored somewhere in a memory section. Figure 1 shows how this bitstream is structured:

The bitstream begins with a token, the preamble, which indicates the beginning of the header section that contains global information concerning the whole configuration cycle. This is followed by the configuration information for the core cells and by the I/O configuration.

Figure 1. Bitstream Structure



8-Bit Microcontroller with Flash

Application Note

0508A

Figure 2. Connecting an Atmel FPGA with the AT89C51.

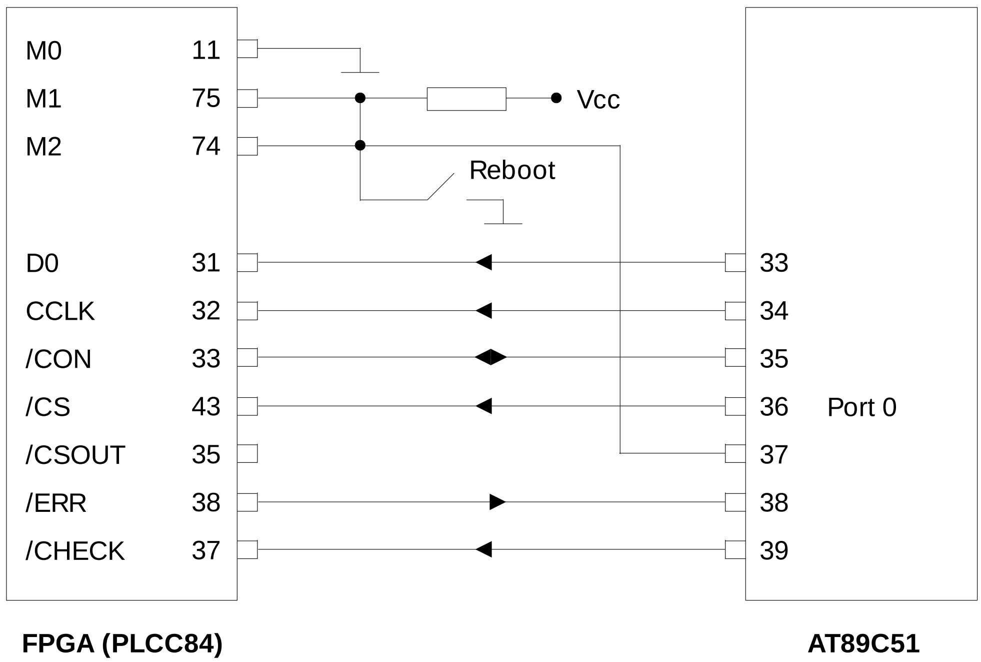
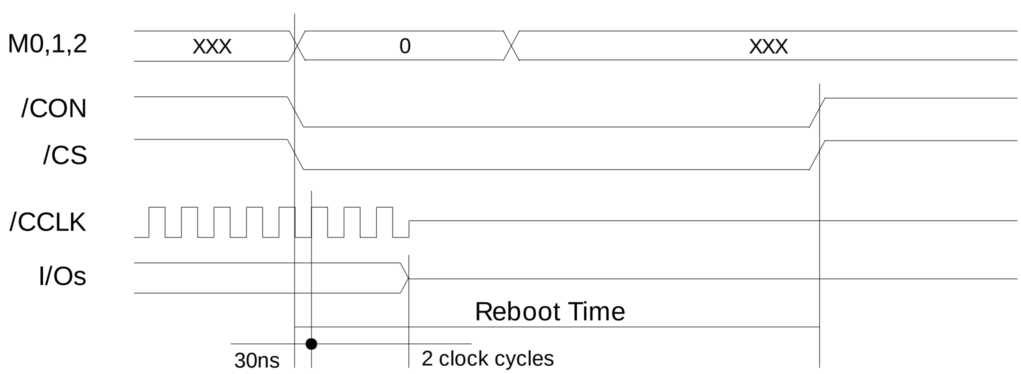


Figure 3. FPGA Reset Timing



The postamble indicates the end of the bitstream.

These data are simply some data bytes that are sent one after another to the FPGA as a text file is sent to a printer. This is done in a serial or parallel fashion by implementing a transfer protocol that is not much different from the transfer protocol of a printer port.

The connections between an FPGA and a microcontroller for serial data transport are shown in Figure 2. Only 7 lines are used enabling full control of the configuration cycle.

The microcontroller can force a complete reset of the whole FPGA by applying the mode 0 to the mode control lines. The state entered by the FPGA is the same state it will enter after power-up. All I/Os are in tristate mode. The exact timing is shown in Figure 3.

After triggering the reboot cycle, the FPGA will pull the /CON line to low. The microcontroller can check this line and detect the end of the cycle.

After a reboot cycle /CS and /CON are held high for two clock cycles. The microcontroller can then select an FPGA

Figure 4. FPGA Configuration Cycle Timing

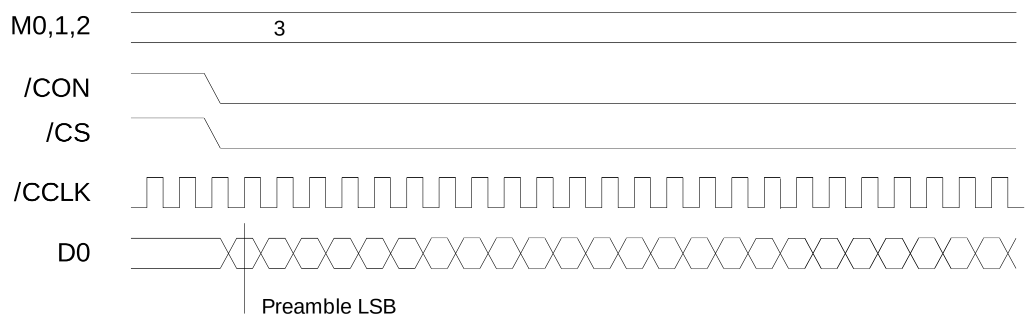
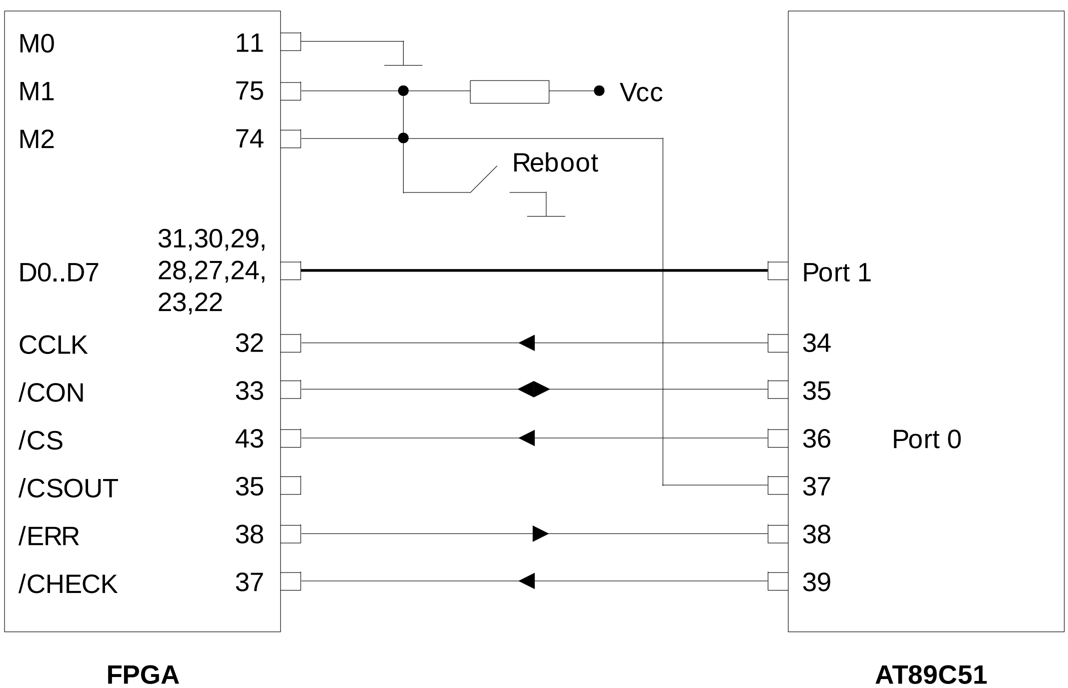


Figure 5. Parallel Data Transfer



through the /CS line for configuration and start the configuration cycle by pulling /CON low. With each clock pulse on the CCLK line one bit of the bitstream transfers to the FPGA. The first transferred bit is the LSB of the preamble, the last transferred bit is the MSB of the postamble. To ensure the correct function of the internal state machine of the FPGA, 24 clock pulses are applied before and after a configuration cycle, for correctly entering the standby state between two configurations. These clock pulses are applied before /CON is pulled low and after /CON goes high. With the transition from low to high the FPGA indicates that the configuration cycle has ended. The exact timing is shown in Figure 4.

This is the configuration mode 3 that is used for configuring several cascaded FPGAs. In this case all lines are brought to all FPGAs in parallel, except for the /CS lines. These connect to the /CSOUT pins of the preceding FPGAs in the chain. Only one bitstream that contains several configurations is needed to serve all FPGAs. The postambles are replaced by preambles to separate them. When the first FPGA in the chain sees a new preamble instead of a postamble, it will activate ist /CSOUT pin. The

next FPGA in the chain is ready to accept the new configuration information, and so on.

Another possibility is to connect all /CS lines to the microcontroller so that it can select the next FPGA to be configured. All other lines are brought to all FPGAs in parallel. In this case normal bitstreams are sufficient, not the special bitstream that is explicitly generated and contains several preambles as described above.

If an error occurs during configuration, the microcontroller can detect this through the /ERR line. When the error is detected from the FPGA, e.g., an invalid preamble, this line will be pulled low to indicate the error to the microcontroller. When using several FPGAs the error lines can all be connected since this is an open collector output that can be WIRE-OR'ed. The microcontroller asserts the /CHECK line to determine whether the FPGA is reconfigured. It also compares the bitstream it receives with the information that is already stored in the configuration memory within the FPGA. In the case of a difference the /ERR line is asserted.

If only reconfiguration of the FPGA without wanting error detection, the number of lines are reduced to three. /CS is tied to GND so that the FPGA is always selected. The microcontroller only has to provide the signals /CON, the configuration clock CCLK, and the data line. This solution uses the least board space.

With parallel data transfer, as shown in Figure 5, 8 data lines instead of one are used to transfer one data byte instead of one data bit per clock cycle. The configuration time is therefore much shorter. For these data lines, the normal data bus of the microcontroller is used, and the data transfer is controlled through the /WR signal of the controller. Reconfiguration of the FPGA is just like writing to an external RAM.

For a system where reconfiguration of the FPGA makes part of the regular system function this might be the most flexible solution. In this mode 6 as illustrated in the data book several FPGAs are cascaded as well. In this case, all lines except for /CS are brought to all FPGAs in parallel. /CS is connected to /CSOUT of the preceding FPGA in the chain to configure several FPGAs with one bitstream. The microcontroller can control the /CS pins to select the FPGA to be reconfigured as in a memory-mapped approach. The exact timing is shown in Figure 6.

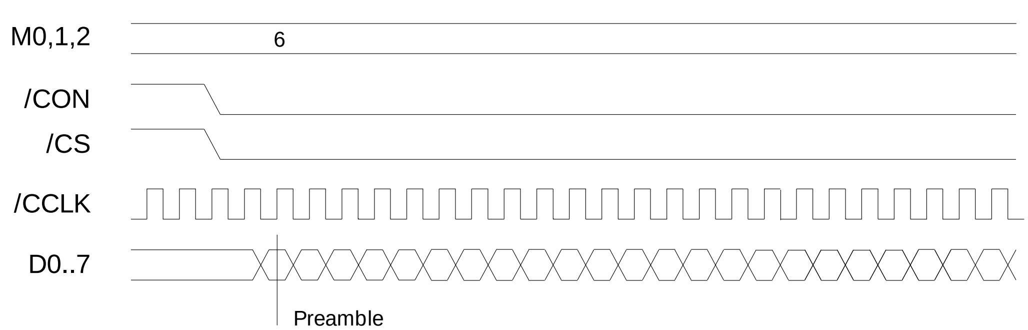
Options for Storing Configuration Data

For storing the configuration data within the system there are different possibilities, each having its advantages and disadvantages.

The first possibility consists of using the internal memory of a controller to store the configuration information. At first glance this might look like a waste but it offers distinctive advantages that enable certain applications. The microcontroller AT89C51 has an internal memory of 4 Kbytes, so that a full configuration of the AT6002 and an additional 1.4 Kbytes of program are stored. In this case only two chips make up the whole system which saves board space. The configuration data is also protected by the lock bits of the controller preventing it from being reverse-engineered.

Another possibility is to store the data in a parallel flash or EPROM memory that is handled by the microcontroller. If a flash memory is used, the configurations are changed through the serial link of the microcontroller simply by downloading a new configuration into the flash memory. Depending on the size of the memory, many configurations are stored in the system. For example, for a 5000 gate FPGA, the AT6005, requires only 8 Kbytes of configuration data. The address space of an AT89C51 controller is 64 Kbytes of data memory. This amounts to eight

Figure 6. FPGA Select Timing



full configurations (or 40000 gates) or, more partial configurations where the bitstream is much smaller depending on how much is changed in a cycle.

The software controlling the FPGA configuration resides within the internal memory of the microcontroller and only needs to know the start address of the bitstreams. It can detect the end itself by searching for the postamble. Another possibility might be a control program that after power-up searches the whole data memory for preambles (for bitstreams). Then the controller receives the command to download the second configuration into the third FPGA by its serial interface, and downloads this configuration into the FPGA, and so on. To control this approach, careful system planning is necessary in order not to destroy a working function.

When using parallel memories instead of serial memories, the configuration is done very fast. Serial memories are more space-efficient, but slower. In space-sensitive applications they can be a solution. There are serial memories that are connected to an FPGA directly. They allow only one configuration, and they are costly. When using a microcontroller, standard serial memories are used that are cheaper and store more than one configuration. For example, if an AT24C64 serial memory with 1^2C bus interface is used, more than 3 full configurations for the AT6002 is stored.

Software Examples

Provided that a configuration cycle has finished with the FPGA releasing the /CON line, a subroutine for transferring a configuration bitstream to a FPGA is relatively simple. In the case of parallel data transfer it is seen as the following:

```
config:     MOV DPTR, #200H                 ;start address bitstream
loopl:      MOVX A, @DPTR                   ;load byte of bitstream
            MOV P1, A                       ;output at port 1
            SETB P0.5                       ;one clock pulse high
            CLR P0.5                        ;clock low
            INC DPTR                        ;next byte
            JNB P0.1, error                 ;ERR is low?
            JNB P0.4, loopl                 ;if CON is low, continue
            RET                             ;configuration finished
error:      do something. . .
```

In the case of serial data transfer a byte of the bitstream has to be converted from parallel to serial, but this is done with ROTATE instructions:

```
            RRC A                           ;first bit to carry
            MOV P1.6, C                     ;output carry as data bit
            RRC A                           ;second bit to carry
            MOV P1.6, C                     ;output carry as data bit
            RRC A                           ;third bit to carry
            MOV P1.6, C                     ;output carry as data bit
            . . .
```

A similar approach is applied in the case of a serial memory containing the configuration bitstreams. The microcontroller assembles them to a byte. When configuring in serial mode, this bit is handed over directly to the FPGA.

When storing several configurations in the address space of a microcontroller, it is complicated and error-prone to change the start addresses of the bitstreams within the microcontroller program. Another possibility is shown in the following code example: a table of 8 start addresses is created and the microcontroller searches the address space for bitstreams. Each time he finds a beginning, the start address is stored in the table. With this approach several configurations are stored at variable locations. In this example it is assumed that every bitstream begins with the preamble and the control register content is 00 hex (this is the second byte of the bitstream). This program is simply to show the principles applying. It is also assumed that an AT89C51 with 4 Kbytes internal memory is used and the bitstreams are stored starting with address 1024 (0400 hex).

```
CONTABLE    db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                                              ;space for 8 16-bit pointers
NUM_CONFIG  db 0                              ;number of configuration
PREAMBLE    EQU #10110010B
POSTAMBLE   EQU #01001101B
CONTROLREG  EQU #00000000B                    ;adjust to desired value
detect:     MOV DPTR, #0400H                  ;start searching at 0400
            MOV R0, NUM_CONFIG                ;actual config address is held in R0
loopl:      MOV A, @DPTR                      ;load byte
            INC DPTR                          ;next byte
            MOV R1, DPH                       ;load high byte DPTR
            CJNE R1, #10H, cont1              ;continue, if address below #1000H (that is,
                                              ;#4096D for a 89C51 that has 4KB of internal
                                              ;memory)
            AJMP overrun                      ;else stop searching (end of memory)
cont1:      CJNE A, PREAMBLE, loop1           ;preamble found?
            MOV A, DPL,                       ;save DPTR low byte
            MOV @R0, A
            INC R0
            MOV A, DPH                        ;save DPTR high byte
            MOV @R0, A
            INC R0                            ;next pointer location
loop2:      MOV A, @DPTR                      ;continue searching
            INC DPTR                          ;next byte
            MOV R1, DPH                       ;test end of memory
            CJNE R1, #10H, cont2
            AJMP overrun
cont2:      CJNE A, CONTROLREG, loop2         ;control reg found?
loop3:      MOV A, @DPTR                      ;yes, continue
            INC DPTR                          ;next byte
            CJNE A, POSTAMBLE, loop3          ;postamble found?
            INC DPTR                          ;continue searching
            MOV R1, DPH                       ;text end of memory
            CJNE R1, #10H, cont3
            AJMP overrun
            AJMP loop1                        ;for preamble
overrun     RET                               ;end of subroutine
```

When it is known how many configurations are stored in the memory and where they are, they can be transferred to the FPGA. for example, through commands received with the serial interface of the microcontroller.

A similar program can be written for storing the configuration data in serial memories, but here the bits have to be assembled to bytes first in order to search for the preamble.

Encryption and Security

SRAM-based FPGAs always receive their configuration from the outside. Besides all advantages this offers, e.g., reconfigurability or testability, there can be problems with security and protection of the design. When the configuration is stored in a serial or parallel memory that is read directly by the FPGA, this memory can be copied. In this case no protection is available.

The problem is only half as difficult as it seems because simply copying the configuration is not the whole job. This can only be used to copy the system, but the logic function of the FPGA is very difficult to deduce from the bitstream. The relation between a certain bit in the bitstream and the function it controls is very difficult to determine. Therefore, the circuit realized with the FPGA is very difficult to reverse-engineer.

When using a microcontroller to configure the FPGA, additional security mechanisms are implemented. The bitstream can be encrypted before storing it in the system memory so that the microcontroller decrypts the bitstream before sending it to the FPGA. The key is hidden within the microcontroller or with external means, e.g., a smartcard or identification number.

Even with very basic operations a high degree of security is reached. For example, if all bits of the bitstream are inverted the configuration bitstream is useless for the FPGA. Another way is to exchange some bytes with others through a table. This is a very easy and therefore fast operation that will slightly slow down the configuration process and will result in a high level of protection.

Using the option indicated above (storing the configuration information within the internal memory of the controller) has other advantages. With the lock bits of the controller access to the memory can be inhibited even when the microcontroller is put on a programmer. With the chip-erase function of the Atmel microcontrollers, the whole memory array can be erased in 10ms when the part of the system is accessed, e.g., by opening the case or entering a wrong identification number three times. This also works when the configuration is not stored within the microcontroller, but only the key number is stored.

There are still weak points in the system. These are made up by the data and control lines between the FPGA and the microcontroller. They are sampled with a logic analyzer and the configuration information is extracted from the timing diagram. This is difficult, but not impossible. One needs to know the parts that are used in the system; the right key or identification number, and a running system for analyzing it. Only then the configuration for one given moment is known. It does not infer that the system can be copied. If partial reconfiguration is used, the design can be partitioned in two or more parts. The major part is transferred unencrypted and some few cells of central importance are transferred at another point of time or from the outside. A system that changes itself frequently is much harder to copy or reverse-engineer. Other tricks such as custom-marking the FPGA (so it is thought to be an ASIC), additional power and ground pins help to disguise the identity of the used part. By implementing all these methods, the process of copying the design is complicated, but there is no absolute security.

Conclusion

The following table shows the different options for parallel or serial configurations in conjunction with parallel or serial configuration storage. The given configuration times are for full configuration of an AT6005 without encryption. An AT89C51 microcontroller with a clock frequency of 24 MHz is used. For assessing the necessary board space, it was assumed that the microcontrollers are used with QFP packages and the memories are used in SOIC or TSOP packages.

Connection to FPGA	Serial	Serial	Parallel	Parallel
External Memory	Serial	Parallel	Serial	Parallel
Space Requirements	$199mm^2$	$329mm^2$	$199mm^2$	$329mm^2$
Configuration Time	93ms	61ms	61ms	30ms

The space requirements are mainly determined by the chosen memory. It is difficult to assess the board space required by parallel or serial wiring. Either one will be determined by application requirements, that is, fast reconfiguration or small space. Configuration time is more dependent on the connection between the controller and the FPGA; the memory connection is not as important.

It is obvious that controlling the configuration of FPGAs with the help of microcontrollers is implemented very easily. When a controller is already in use within the system, only one additional port is required, and some space in the flash memory that might already be in the system as well. Flexibility in the design is increased and additional features can easily be implemented.